

**NAME:** SAMHITHA D  
**USN:** 1BM20CS141

MACHINE LEARNING LAB OBSERVATION

Date: 1-04-2023

## Lab 1: Exploring Datasets

## IRIS DATASET:

- Features in the Iris dataset:

1. sepal length in cm
  2. sepal width in cm
  3. petal length in cm
  4. petal width in cm

- Target classes to predict:

1. Iris Setosa
  2. Iris Versicolour
  3. Iris Virginica

```
In [8]: from sklearn.datasets import load_iris  
iris=load_iris()
```

```
In [9]: print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],  
[4.9, 3. , 1.4, 0.2],  
[4.7, 3.2, 1.3, 0.2],  
[4.6, 3.1, 1.5, 0.2],  
[5. , 3.6, 1.4, 0.2],  
[5.4, 3.9, 1.7, 0.4],  
[4.6, 3.4, 1.4, 0.3],  
[5. , 3.4, 1.5, 0.2],  
[4.4, 2.9, 1.4, 0.2],  
[4.9, 3.1, 1.5, 0.1],  
[5.4, 3.7, 1.5, 0.2],  
[4.8, 3.4, 1.6, 0.2],  
[4.8, 3. , 1.4, 0.1],  
[4.3, 3. , 1.1, 0.1],  
[5.8, 4. , 1.2, 0.2],  
[5.7, 4.4, 1.5, 0.4],  
[5.4, 3.9, 1.3, 0.4],  
[5.1, 3.5, 1.4, 0.3],  
[5.7, 3.8, 1.7, 0.3],
```

```
In [5]: type(iris)
```

Out[5]: function

```
In [12]: iris.keys()
```

```
Out[12]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [13]: iris

```
[4., 7., 3., 2., 1., 6., 0., 2.],
[4., 8., 3., 1., 1., 6., 0., 2.],
[[5., 4., 3., 4., 1., 5., 0., 4.],
[5., 2., 4., 1., 1., 5., 0., 1.],
[5., 5., 4., 2., 1., 4., 0., 2.],
[4., 9., 3., 1., 1., 5., 0., 2.],
[5., , 3., 2., 1., 2., 0., 2.],
[[5., 5., 3., 5., 1., 3., 0., 2.],
[4., 9., 3., 6., 1., 4., 0., 1.],
[4., 4., 3., , 1., 3., 0., 2.],
[5., 1., 3., 4., 1., 5., 0., 2.],
[5., , 3., 5., 1., 3., 0., 3.],
[[4., 5., 2., 3., 1., 3., 0., 3.],
[4., 4., 3., 2., 1., 3., 0., 2.],
[5., , 3., 5., 1., 6., 0., 6.],
[5., 1., 3., 8., 1., 9., 0., 4.],
[4., 8., 3., , 1., 4., 0., 3.],
[5., 1., 3., 8., 1., 6., 0., 2.],
```

```
In [17]: print(iris['target_names'])  
['setosa' 'versicolor' 'virginica']
```

```
In [20]: n_samples,n_features=iris.data.shape  
print("no.of samples:",n_samples)  
print("no.of features:",n_features)  
  
no.of samples: 150  
no.of features: 4
```

```
In [28]: iris.data[[12,26,89,114]]
```

```
Out[28]: array([[4.8, 3. , 1.4, 0.1],  
                 [5. , 3.4, 1.6, 0.4],  
                 [5.5, 2.5, 4. , 1.3],  
                 [5.8, 2.8, 5.1, 2.4]])
```

```
In [29]: print(iris.data.shape)
```

```
(150, 4)
```

```
In [31]: print(iris.target.shape)
```

```
(150,)
```

```
In [32]: import numpy as np  
np.bincount(iris.target)
```

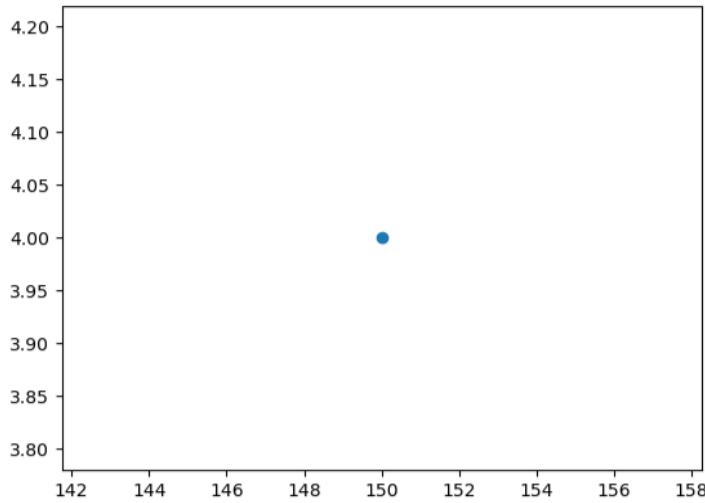
Scattered graph for samples vs features.

```
In [32]: import numpy as np  
np.bincount(iris.target)
```

```
Out[32]: array([50, 50, 50], dtype=int64)
```

```
In [42]: import matplotlib.pyplot as plt  
plt.scatter(n_samples,n_features)
```

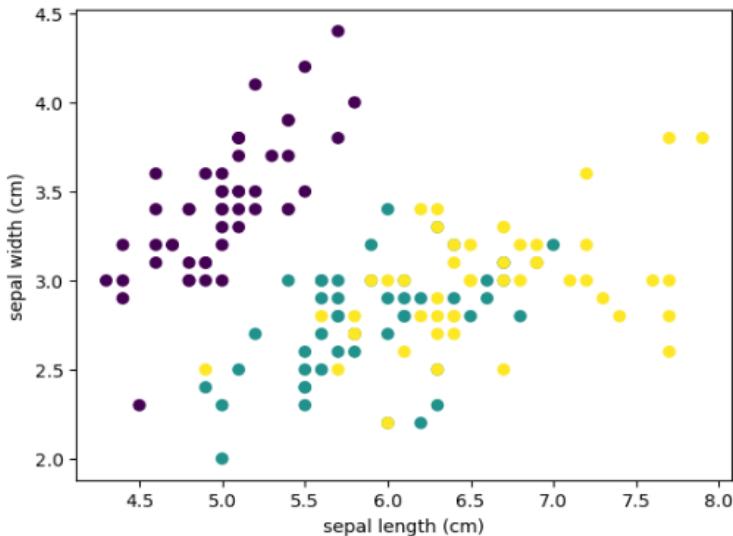
```
Out[42]: <matplotlib.collections.PathCollection at 0x1d1c8c45550>
```



Scattered graph: with first two features( septal width vs septal length)  
The three colors represents three different classes respectively.

```
In [47]:
```

```
features = iris.data.T  
  
plt.scatter(features[0], features[1],  
           c=iris.target)  
plt.xlabel(iris.feature_names[0])  
plt.ylabel(iris.feature_names[1]);
```



```
In [49]: iris.data[[1,2,3,4,5]]
```

```
Out[49]: array([[4.9, 3., 1.4, 0.2],  
                 [4.7, 3.2, 1.3, 0.2],  
                 [4.6, 3.1, 1.5, 0.2],  
                 [5., 3.6, 1.4, 0.2],  
                 [5.4, 3.9, 1.7, 0.4]])
```

## WINE DATASET:

```
In [51]: from sklearn.datasets import load_wine  
wine=load_wine()
```

```
In [52]: print(wine)
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,  
                1.065e+03],  
                [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,  
                1.050e+03],  
                [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
```

```
In [57]: wine.data
```

```
Out[57]: array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,  
                1.065e+03],  
                [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,  
                1.050e+03],  
                [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,  
                1.185e+03],  
                ...,  
                [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,  
                8.350e+02],  
                [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,  
                8.400e+02],  
                [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,  
                5.600e+02]])
```

```
In [58]: wine.keys()
```

```
Out[58]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

```
In [60]: print(wine['target_names'])
```

```
['class_0' 'class_1' 'class_2']
```

```
In [9]: print(wine['feature_names'])

['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proa
nthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

```
In [11]: import numpy as np
np.bincount(wine.target)
```

```
Out[11]: array([59, 71, 48], dtype=int64)
```

Date: 15/04/2023

Lab 2: FIND-S ALGORITHM FOR ENJOY SPORT:

**Program 2 –** Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file Data set:Enjoysport

a. Enjoysport

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Algorithm:

initialize h to the most specific hypothesis in H  $h-(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

1. First training example  $X_1 = < \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} >$ . EnjoySport=+ve Observing. The first training example, it is clear that hypothesis h is too specific. None of the " $\emptyset$ " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example  $h_1 = < \text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same} >$ .
2. Consider the second training example  $x_2 < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Warm}, \text{Same} >$ . EnjoySport+ve. The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. Now  $h_2 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} >$
3. Consider the third training example  $x_3 < \text{Rainy}, \text{Cold}, \text{High}, \text{Strong}, \text{Warm}, \text{Change} >$ . EnjoySport ve. The FIND-S algorithm simply ignores every negative example. So the hypothesis remain as before, so  $h_3 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same} >$
4. Consider the fourth training example  $x_4 < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Change}, \text{EnjoySport} >$  +ve. The fourth example leads to a further generalization of h as  $h_4 = < \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? >$
5. So the final hypothesis is  $< \text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ? >$

```

import csv
a = []
with open('kaggle/input/enjoysport/enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
print(a)

print(m 'The total no. of training instances are : ', len(a))
num_attributes = len(a[0]) - 1
print("The initial hypothesis is : ")
hypothesis = ['0'] * num_attributes
print(hypothesis)
for i in range(0, len(a)):
    if a[i][num_attributes] == 'yes':
        for j in range(0, num_attributes):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
print("The hypothesis for the training instance ", i, " is : ")
print(hypothesis)

```

OUTPUT: [ 'sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast',  
       'enjoysport'], [ 'sunny', 'warm', 'normal', 'strong', 'warm',  
       'same', 'yes'], ...

The total number of training instances are : 5

The initial hypothesis is :  
       ['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is :  
       ['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 2 is :  
       ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

--> 3 is :  
       ['sunny', 'warm', '?', 'strong', 'warm', 'same']

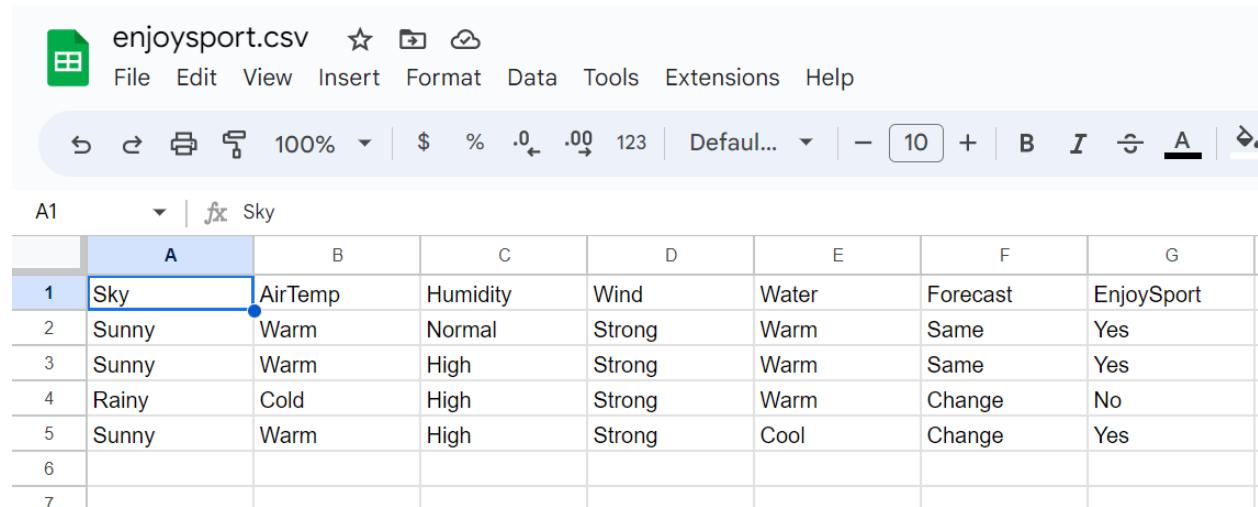
--> 4 is :  
       ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The maximally specific hypothesis for the training instance  
       is  
       ['sunny', 'warm', '?', 'strong', '?', '?']

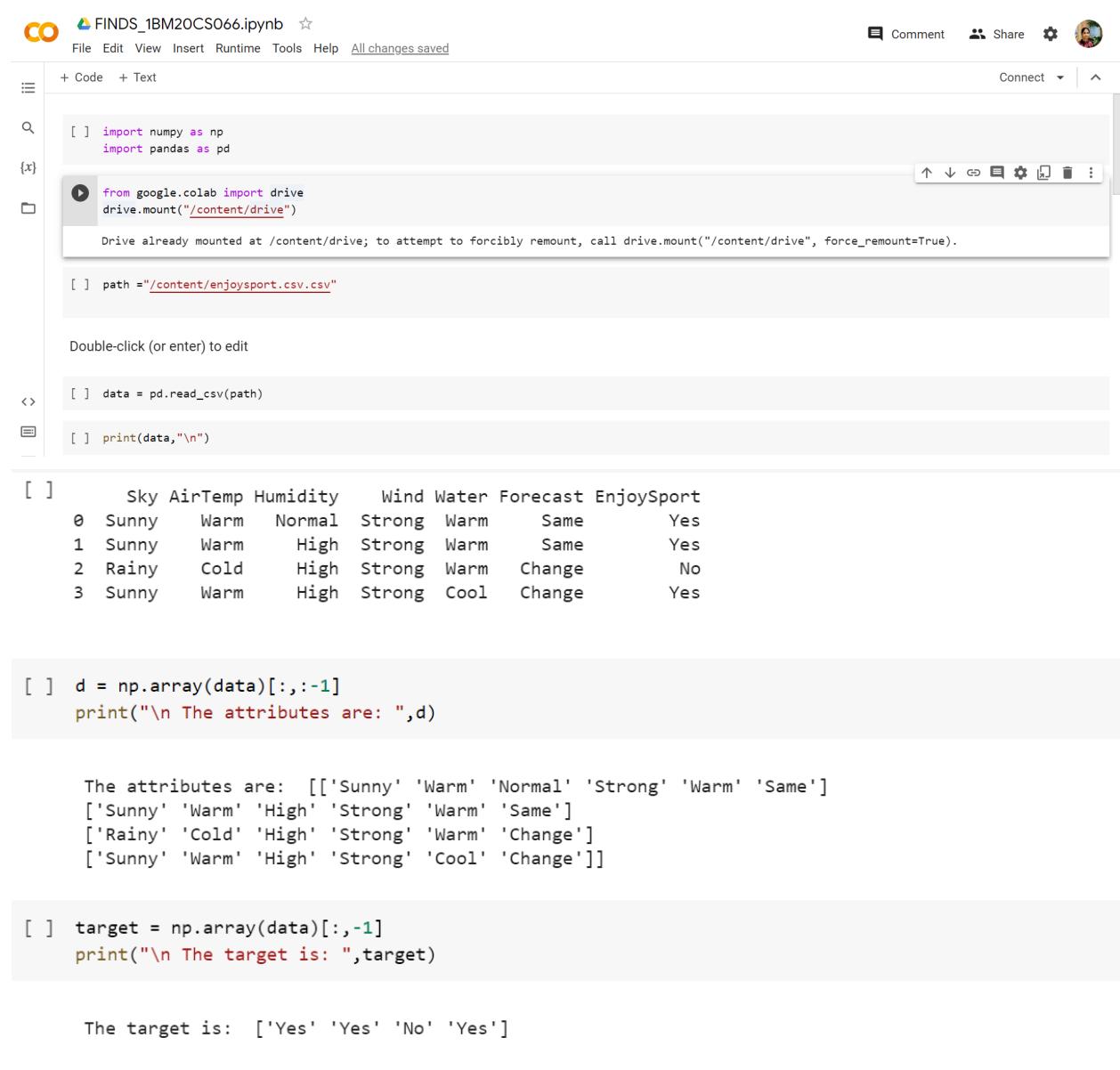
/ Ans. w/ 2

the training  
       examples  
       seen now

## CREATING CSV FILE:



	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							



```
[ ] import numpy as np
import pandas as pd
from google.colab import drive
drive.mount("/content/drive")
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[ ] path ="/content/enjoysport.csv.csv"
Double-click (or enter) to edit
[ ] data = pd.read_csv(path)
[ ] print(data,"\n")
[ ] Sky AirTemp Humidity Wind Water Forecast EnjoySport
0 Sunny Warm Normal Strong Warm Same Yes
1 Sunny Warm High Strong Warm Same Yes
2 Rainy Cold High Strong Warm Change No
3 Sunny Warm High Strong Cool Change Yes
[ ] d = np.array(data)[:, :-1]
print("\n The attributes are: ",d)

The attributes are: [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

[ ] target = np.array(data)[:, -1]
print("\n The target is: ",target)

The target is: ['Yes' 'Yes' 'No' 'Yes']
```

```
[ ] def finds(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

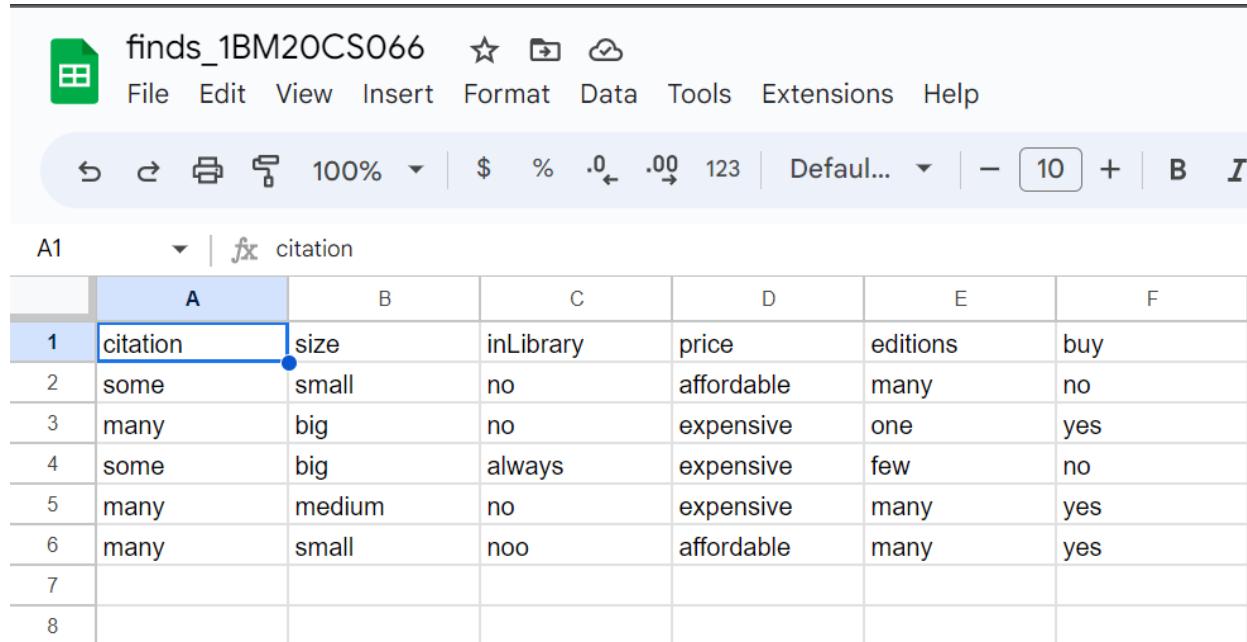
print("\n The final hypothesis is:",finds(d,target))
```

The final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']

## SECOND DATASET: FIND-S ALGORITHM

example	citations	size	inLibrary	price	editions	buy
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

## CREATING CSV FILE



A	B	C	D	E	F
1	citation	size	inLibrary	price	editions
2	some	small	no	affordable	many
3	many	big	no	expensive	one
4	some	big	always	expensive	few
5	many	medium	no	expensive	many
6	many	small	noo	affordable	many
7					
8					

```
▶ import numpy as np  
import pandas as pd
```

```
[ ] from google.colab import drive  
drive.mount("/content/drive")
```

Mounted at /content/drive

```
[ ] path ="/content/finds_1BM20CS066 - Sheet1.csv"
```

```
[ ] data = pd.read_csv(path)
```

```
[ ] print(data, "\n")
```

	citation	size	inLibrary	price	editions	buy
0	some	small	no	affordable	many	no
1	many	big	no	expensive	one	yes
2	some	big	always	expensive	few	no
3	many	medium	no	expensive	many	yes
4	many	small	noo	affordable	many	yes

```
[ ] d = np.array(data)[:, :-1]  
print("\n The attributes are: ",d)
```

The attributes are: [['some' 'small' 'no' 'affordable' 'many']  
['many' 'big' 'no' 'expensive' 'one']  
[['some' 'big' 'always' 'expensive' 'few']]  
[['many' 'medium' 'no' 'expensive' 'many']]  
[['many' 'small' 'noo' 'affordable' 'many']]

```
▶ target = np.array(data)[:, -1]  
print("\n The target is: ",target)
```

The target is: ['no' 'yes' 'no' 'yes' 'yes']

+ Code + Text

```
[ ] def find_s(d,target):  
    for i, val in enumerate(target):  
        if val=='yes':  
            hypothesis=d[i].copy()  
            break  
  
    for i, var in enumerate(d):  
        if target[i]=="yes":  
            for x in range(len(hypothesis)):  
                if var[x]!=hypothesis[x]:  
                    hypothesis[x]='?'  
                else:  
                    pass  
  
    return hypothesis  
  
print("The Hypothesis is",find_s(d,target))
```

The Hypothesis is ['many' '?' '?' '?' '?']

**DATE:** 15/04/2023

### **LAB 3: CANDIDATE- ELIMINATION- ENJOY SPORT**

**Program 3:** For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.  
Data set: Enjoy sport

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

#### **ALGORITHM:**

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

    if attribute\_value == hypothesis\_value:

        Do nothing

    else:

        replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

    Make generalize hypothesis more specific.

? match with all the alt. expected outcome  
 : inconsistent.  
 so write all hypo. which are consistent till now.  
 To do that, consider one ? at a time.

CODG :

```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])
print(target)
```

```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific-h & general-h")
    print(specific_h)
    general_h = ['?' for i in range(len(specific_h))]
    for i in range(len(specific_h)):
        if target[i] == "yes":
            for x in range(len(concepts)):
                if concepts[x][i] == specific_h[i]:
                    specific_h[i] = '1'
                    general_h[i] = '?'
                    print(specific_h)
        else:
            for x in range(len(concepts)):
                if concepts[x][i] != specific_h[i]:
                    specific_h[i] = '0'
                    general_h[i] = '?'
                    print(specific_h)
```

Date: 12/04/23  
 Page:

```
general_h[x] = specific_h[x]
else:
    general_h[x] = '?'
print("steps of C & A go", i+1)
print(specific_h)
print(general_h)
indices = [i for i, val in enumerate(general_h) if val == '?']
val = ['?', '?', '?', '?', '?', '?', '?']
for i in indices:
    general_h.remove('?')
    general_h.append(val[i])
print(specific_h, general_h)
s_final, g_final = learn(concepts, target)
print("Final Specific:", s_final, sep="\n")
print("Final General:", g_final, sep="\n")
```

o/p:

Final specific-h:  
 ['sunny', 'warm', '?', 'strong', '?', '?']

Final general-h: ['?', '?', '?', '?', '?', '?', '?']  
 ['?', '?', '?', '?', '?', '?', '?']

12-4-23

## Dataset

origin	Manufacturer	color	decade	type	descriptu
Japan	Honda	blue	1980	economy	yes
Japan	Toyota	green	1970	sport	no
Japan	Toyota	blue	1970	economy	yes
US A	Toyota	green	1980	economy	no
Japan	Honda	blue	1980	economy	yes

o/p:

Final specific-h:  
[{"japan", "?", "blue", "?", "economy"}]

Final general-h:

[{"japan", ?, ?, ?, ?}, [{"? ?, ?, ?}, {"? ?, ?"}]]

## CREATING CSV FILE:

A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast
2	Sunny	Warm	Normal	Strong	Warm	Same
3	Sunny	Warm	High	Strong	Warm	Same
4	Rainy	Cold	High	Strong	Warm	Change
5	Sunny	Warm	High	Strong	Cool	Change
6						
7						

```
[ ] import numpy as np
import pandas as pd

[ ]
from google.colab import drive
drive.mount('/content/drive')

[ ]
data = pd.DataFrame(data=pd.read_csv('/content/enjoysport.csv.csv'))

[ ]
print(data,"\n")

   Sky  AirTemp  Humidity    Wind  Water  Forecast EnjoySport
0  Sunny      Warm  Normal  Strong   Warm     Same      Yes
1  Sunny      Warm    High  Strong   Warm     Same      Yes
2  Rainy     Cold    High  Strong   Warm  Change      No
3  Sunny      Warm    High  Strong   Cool  Change      Yes
```

```
[ ] concepts = np.array(data.iloc[:,0:-1])

[ ]
print(concepts)

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

[ ] target = np.array(data.iloc[:,-1])
print(target)

['Yes' 'Yes' 'No' 'Yes']

[ ] import csv
```

```

with open("/content/enjoysport.csv.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

    print("\nStep " + str(data.index(i)) + " of Candidate Elimination Algorithm")
    print(specific)
    print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
                gh.append(i)
                break

    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)

```

```

[ ] def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [[ "?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("Step 0:")
    print("Specific Hypothesis: ", specific_h)
    print("General Hypothesis: ", general_h)
    print("-----")
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    print("Step", i+1, ":")
    print("Specific Hypothesis: ", specific_h)
    print("General Hypothesis: ", general_h)
    print("-----")
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")

```

## SECOND DATASET:

example	citations	size	inLibrary	price	editions	buy
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

## CREATING CSV FILE:

File Edit View Insert Format Data Tools Extensions Help

100% | \$ % .0 .00 123 | Default... | - | + | B I

A1 | fx citation

	A	B	C	D	E	F
1	citation	size	inLibrary	price	editions	buy
2	some	small	no	affordable	many	no
3	many	big	no	expensive	one	yes
4	some	big	always	expensive	few	no
5	many	medium	no	expensive	many	yes
6	many	small	noo	affordable	many	yes
7						
8						

from google.colab import drive  
drive.mount('/content/drive')

Mounted at /content/drive

```
[ ] import numpy as np  
import pandas as pd
```

```
[ ] data = pd.DataFrame(data=pd.read_csv('/content/finds_1BM20CS066 - Sheet1.csv'))  
print(data, "\n")
```

	citation	size	inLibrary	price	editions	buy
0	some	small	no	affordable	many	no
1	many	big	no	expensive	one	yes
2	some	big	always	expensive	few	no
3	many	medium	no	expensive	many	yes
4	many	small	noo	affordable	many	yes

```
[ ] concepts = np.array(data.iloc[:, 0:-1])  
print("The attributes are: ", concepts)
```

The attributes are: [['some' 'small' 'no' 'affordable' 'many']  
 ['many' 'big' 'no' 'expensive' 'one']  
 ['some' 'big' 'always' 'expensive' 'few']  
 ['many' 'medium' 'no' 'expensive' 'many']  
 ['many' 'small' 'noo' 'affordable' 'many']]

```
[ ] target = np.array(data.iloc[:, -1])  
print("\n The target is: ", target)
```



**Program 4:** Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

#### ALGORITHM:

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target\_attribute in Examples
- Otherwise Begin
  - $A \leftarrow$  the attribute from Attributes that best\* classifies Examples
  - The decision attribute for Root  $\leftarrow A$
  - For each possible value,  $v_i$ , of A,
  - Add a new tree branch below Root, corresponding to the test  $A = v_i$
  - Let  $Examples_{v_i}$  be the subset of Examples that have value  $v_i$  for A
  - If  $Examples_{v_i}$  is empty
  - Then below this new branch add a leaf node with label = most common value of Target\_attribute in Examples
  - Else below this new branch add the subtree  $ID3(Examples_{v_i}, Target\_attribute, Attributes - \{A\})$
  - End
- Return Root

CLASSEMALE  
Date \_\_\_\_\_  
Page \_\_\_\_\_

ID3

```

import pandas as pd
import math
import numpy as np

data = pd.read_csv("path.csv")
feature_ids = [feat for feat in data]
features = data.drop(["answer"])

```

```

class Node:
    def __init__(self):
        self.children = []
        self.value = None
        self.isleaf = False
        self.pred = None

    def entropy(examples):
        pos = 0.0
        neg = 0.0
        for row in examples.itertuples():
            if row["answer"] == "yes":
                pos += 1
            else:
                neg += 1
        if pos == 0.0 or neg == 0.0:
            return 0.0
        else:
            p = pos / (pos + neg)
            n = neg / (pos + neg)
            return -(p * math.log(p, 2) + n * math.log(n, 2))

    def info_gain(examples, attr):
        unique = np.unique(examples[attr])
        gain = entropy(examples)

```

```

for v in unique:
    subdata = examples[examples[attr] == v]
    subent = entropy(subdata)
    gain -= (float(len(subdata)) / float(len(examples))) * subent
return gain

def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = None
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    unique = np.unique(examples[max_feat])
    for v in unique:
        subdata = examples[examples[max_feat] == v]
        if entropy(subdata) == 0.0:
            new_node = Node()
            new_node.isLeaf = True
            new_node.value = v
            new_node.pred = np.unique(subdata["answer"])
            root.children.append(new_node)
        else:
            dummyNode = Node()
            dummyNode.value = v
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

```

Date \_\_\_\_\_  
Page \_\_\_\_\_

return root

```

examples) * solve
def printTree(root, depth=0):
    for i in range(depth):
        print("  ", end=" ")
    print("(" + str(root.value) + ", " + str(root.end) + ")")
    if root.isleaf:
        print("  leaf")
    else:
        print("  child")
    for child in root.children:
        printTree(child, depth+1)

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isleaf:
                print("Predicted Label for new example", new,
                      "is:", child.value)
                exit()
            else:
                classify(child, new)
                break
    print("No child found for value", new[root.value])

```

= 0

"answer")

root = ID3(data, features)

print("Decision Tree is:")

printTree(root)

print("-----")

new = { "outlook": "sunny", "temperature": "hot",  
 "humidity": "normal", "wind": "strong" }

classify(root, new)

outlook	temperature	humidity	wind	answer
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Q/A:

Decision Tree is:

outlook

overcast → ['yes']

rain

wind

strong → ['no']

weak → ['yes']

sunny

humidity

high → ['no']

normal → ['yes']

Predicted label for new example [ 'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong' ] is: ['yes']



1BM20CS066\_ID3



File Edit View Insert Format Data Tools Extensions Help



100% ▾

\$

%

.0

.00

123

Default... ▾



A1

fix outlook

	A	B	C	D	E
1	outlook	temperature	humidity	wind	play tennis
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no
16					



ID3.ipynb

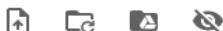


File Edit View Insert Runtime Tools Help All changes saved

Files



+ Code + Text



{x} .. sample\_data 1BM20CS066\_ID3.csv

```
[53] import math
import csv

[55] def load_csv(filename):
    lines=csv.reader(open(filename,"r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

[56] class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children={}
        self.answer=""

[57] def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]

    pos=0
    for y in range(r):
        if data[y][col]==attr[x]:
            if delete:
                del data[y][col]
            dic[attr[x]][pos]=data[y]
            pos+=1

    return attr,dic
```



Disk

84.31 GB available

```
✓ [58] def entropy(s):
    attr=list(set(s))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in s if attr[i]==x])/(len(s)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums
```

```
✓ [59] def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

```
✓ [60] def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]

    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
```

```
✓  def print_tree(node,level):
    if node.answer!="":
        print(" "+*level,node.answer)
        return

        print(" "+*level,node.attribute)
        for value,n in node.children:
            print(" "+*(level+1),value)
            print_tree(n,level+2)
```

```
✓ [62] def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

```
✓ [63]
dataset,features=load_csv("1BM20CS066_ID3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("1BM20CS066_ID3.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
outlook
  rain
    wind
      weak
        yes
        strong
        no
      sunny
        humidity
          high
          no
        normal
        yes
      overcast
        yes
```

```
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:
yes
```

```

The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:
no
```

# PROGRAM 5: Simple linear regression program

Dataset used:

	A	B
1	x	y
2		1
3		2
4		3
5		3.75
6		5
7		2.25

## ALGORITHM:

- The main function to calculate values of coefficients
- Initialize the parameters.
- Predict the value of a dependent variable by giving an independent variable.
- Calculate the error in prediction for all data points.
- Calculate partial derivatives w.r.t  $a_0$  and  $a_1$ .
- Calculate the cost for each number and add them.
- Update the values of  $a_0$  and  $a_1$ .

Linear Regression

```
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd
x = np.linspace(0, 100, 50)
y = x + np.random.normal(2*x + 2, 20)
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.scatter(x, y)

X = x
Y = y
x_mean = np.mean(X)
y_mean = np.mean(Y)
n = len(X)
numerator = 0
denominator = 0
for i in range(n):
    numerator += (X[i] - x_mean) * (Y[i] - y_mean)
    denominator += (X[i] - x_mean) ** 2
b1 = numerator / denominator
b0 = y_mean - (b1 * x_mean)
print(b1, b0)
rmse = np.sqrt(rmse/n)
print(rmse)
```

```
[ ] import numpy as np
import matplotlib.pyplot as plt

[ ] def plot_regression_line(x, y, b):

    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "g")

    plt.xlabel('x CO-EFF')
    plt.ylabel('y CO-EFF')

    plt.show()
```

```
[ ] def estimate_coef(x, y):

    n = np.size(x)

    m_x = np.mean(x)
    m_y = np.mean(y)

    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)
```

```
▶ def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "b",
                marker = "*", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "y")

    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()
```

```

def main():

    x = np.array([1,2,3,4,5])
    y = np.array([1,2,1.3,3.75,2.25])

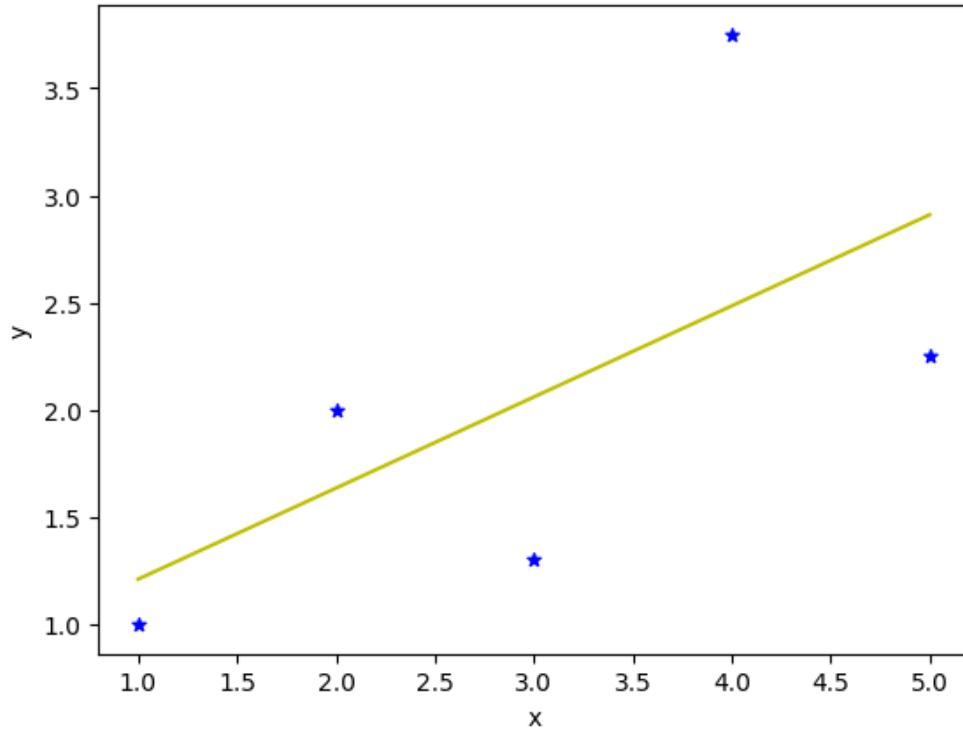
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \
        \nb_1 = {}".format(b[0], b[1]))

    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()

```

Estimated coefficients:  
 $b_0 = 0.785000000000001$   
 $b_1 = 0.4249999999999966$



## Conclusion:

This model is not appropriate for this model. All the points of this dataset are away from the prediction line.

**Program 6: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

**Data set used:**

	A	B
1	outlook	play
2	rainy	Yes
3	sunny	Yes
4	overcast	Yes
5	overcast	Yes
6	sunny	No
7	rainy	Yes
8	sunny	Yes
9	overcast	Yes
10	rainy	No
11	sunny	No
12	sunny	Yes
13	rainy	No
14	overcast	Yes
15	overcast	Yes

**Algorithm:**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Formula for naive bayes classifier is as follows →

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.
4. Test accuracy of the result and visualizing the test set result.

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

Name Boys Classifier

import csv  
import random  
import math

```

def loadCSV(filename):
    lines = csv.reader(open('mnistdata.csv', 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

```

```

def splitdataset(dataset, splitratio):
    trsize = int(len(dataset) * splitratio)
    copy = list(dataset)
    trainset = []
    while len(copy) > trsize:
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

```

```

def separatebyclass(dataset):
    sepa = {}
    for i in range(len(dataset)):
        if dataset[i][-1] not in sepa:
            sepa[dataset[i][-1]] = []
        sepa[dataset[i][-1]].append(dataset[i])
    return sepa

```

```

def mean(numbers):
    return sum(numbers) / float(len(numbers))

```

```

countYes = 0
countNo = 0
probYes = 0
probNo = 0
print("n")
print("target (count probability)")

for x in range(data.shape[0]):
    if data[x, data.shape[1]-1] == "Yes":
        countYes += 1
    if data[x, data.shape[1]-1] == "No":
        countNo += 1

probYes = countYes / total_size
probNo = countNo / total_size

print("Yes", "1t", countYes, "1t", probYes)
print("No", "1t", countNo, "1t", probNo)

prob0 = np.zeros((test.shape[1]-1))
prob1 = np.zeros((test.shape[1]-1))
accuracy = 0
print("n")
print("instance prediction target")

for t in range(test.shape[0]):
    for k in range(test.shape[1]-1):
        count1 = count0 = 0
        for j in range(data.shape[0]):
            if test[t, k] == data[j, k] and
               data[j, data.shape[1]-1] == "No":
                count0 += 1
            if test[t, k] == data[j, k] and
               data[j, data.shape[1]-1] == "Yes":
                count1 += 1
        if test[t, k] == data[j, k] and
           data[j, data.shape[1]-1] == "No":
            prob0[k] = count0 / total
        if test[t, k] == data[j, k] and
           data[j, data.shape[1]-1] == "Yes":
            prob1[k] = count1 / total
    accuracy += 1
print("n")

```

CLASSMATE  
Date 17/05/23  
Page

Joint probability  $P(h)$

$\approx \max \cdot P(V_j) \cdot \pi P(a_i | V_j)$

$$P(V_{NB}) = \frac{P(V_{NB} | yes)}{P(V_{NB} | yes) + P(V_{NB} | no)}$$

prob0 = prob\_nb  
prob1 = prob\_yes  
for i in range (test.shape[1] - 1):  
prob0 = prob\_nb \* prob0[i]  
prob1 = prob\_yes \* prob1[i]  
if prob0 > prob1:  
predict = 'no'  
else:  
predict = 'yes'  
print(t+1, "t", predict, "t", test[t, test.shape[1]-1])  
if predict == test[t, test.shape[1]-1]:  
accuracy += 1  
final\_accuracy = (accuracy / test.shape[0]) \* 100  
print("accuracy", final\_accuracy, "%")  
return

and

metadata, traindata = read\_data('kaggle/input/dataset\_mushroom')  
splitRatio = 0.8  
trainingSet, testSet = splitDataset(metadata, splitRatio)  
training = np.array(trainingSet)  
print('Training set size:', len(training))  
for x in training:  
print(x)  
testing = np.array(testSet)  
print('Test set size:', len(testing))  
for x in testing:  
print(x)  
classify(training, testing)

obj:

The training data set are:

- [Red, 'Sports', 'Domestic', 'Yes']
- [Red, 'Sports', 'Domestic', 'No']
- [Red, 'Sports', 'Domestic', 'No']
- [Yellow, 'Sports', 'Domestic', 'No']
- [Yellow, 'Sports', 'Imported', 'Yes']
- [Yellow, 'SUV', 'Imported', 'No']

The test set are:

- [Yellow, SUV, Imported, Yes]
- [Yellow, SUV, Domestic, No]
- [Red, SUV, Imported, No]
- [Red, SUV, Imported, Yes]

Training data size = 6  
Test data size = 4

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

dataset movie (iris).csv  
(Ratio)

target	count	probability
Yes	3	0.5
No	3	0.5

instance	prediction	target
1	No	Yes
2	No	No
3	No	No
4	Yes	Yes

accuracy 75.0%

1BM20CS066\_NBC.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- ..
- sample\_data
- 1BM20CS066\_NBC.csv

+ Code + Text

```

[7] import numpy as np
import math
import csv
import pdb

[8] def read_data(filename):
    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

[9] def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]

```

```
✓ 0s def classify(data,test):  
  
    total_size = data.shape[0]  
    print("\n")  
    print("training data size=",total_size)  
    print("test data size=",test.shape[0])  
  
    countYes = 0  
    countNo = 0  
    probYes = 0  
    probNo = 0  
    print("\n")  
    print("target      count      probability")  
  
    for x in range(data.shape[0]):  
        if data[x,data.shape[1]-1] == 'Yes':  
            countYes +=1  
        if data[x,data.shape[1]-1] == 'No':  
            countNo +=1  
  
    probYes=countYes/total_size  
    probNo= countNo / total_size  
  
    print('Yes','\t',countYes,'\t',probYes)  
    print('No','\t',countNo,'\t',probNo)  
  
    prob0 =np.zeros((test.shape[1]-1))  
    prob1 =np.zeros((test.shape[1]-1))  
    accuracy=0  
    print("\n")  
    print("instance prediction  target")  
  
    for t in range(test.shape[0]):  
        for k in range (test.shape[1]-1):  
            count1=count0=0  
            for j in range (data.shape[0]):  
                #how many times appeared with no  
                if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='No':  
                    count0+=1  
                #how many times appeared with yes  
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='Yes':  
                    count1+=1  
  
            prob0[k]=count0/countNo  
            prob1[k]=count1/countYes  
  
            probno=probNo  
            probyes=probYes  
            for i in range(test.shape[1]-1):  
                probno=probno*prob0[i]  
                probyes=probyes*prob1[i]  
            if probno>probyes:  
                predict='No'  
            else:  
                predict='Yes'  
  
            print(t+1," \t",predict," \t ",test[t,test.shape[1]-1])  
            if predict == test[t,test.shape[1]-1]:  
                accuracy+=1  
    final_accuracy=(accuracy/test.shape[0])*100  
    print("accuracy",final_accuracy,"%")  
    return
```

```
metadata,traindata= read_data("/content/1BM20CS066_NBC.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```

### output:

```
The Training data set are:
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']
['overcast', 'Yes']
['sunny', 'No']
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']
```

```
The Test data set are:
['rainy' 'No']
['sunny' 'No']
['sunny' 'Yes']
['rainy' 'No']
['overcast' 'Yes']
['overcast' 'Yes']
```

```
training data size= 8
test data size= 6
```

target	count	probability
Yes	7	0.875
No	1	0.125

instance	prediction	target
1	Yes	No
2	Yes	No
3	Yes	Yes
4	Yes	No
5	Yes	Yes
6	Yes	Yes

accuracy 50.0 %

## Program 7:K- means clustering

### Algorithm:

Initialize k means with random values

For a given number of iterations:

Iterate through items:

Find the mean closest to the item by calculating the euclidean distance of the item with each of the means

Assign item to mean

Update mean by shifting it to the average of the items in that cluster

### Dataset:

Kmeans_1BM20CS066.csv X				...
1	Name	Age	Income(\$)	Filter
2	Rob	27	70000	
3	Michael	29	90000	
4	Mohan	29	61000	
5	Ismail	28	60000	
6	Kory	42	150000	
7	Gautam	39	155000	
8	David	41	160000	
9	Andrea	38	162000	
10	Brad	36	156000	
11	Angelina	35	130000	
12	Donald	37	137000	
13	Tom	26	45000	
14	Arnold	27	48000	
15	Jared	28	51000	
16	Stark	29	49500	
17	Ranbir	32	53000	
18	Dipika	40	65000	
19	Priyanka	41	63000	
20	Nick	43	64000	
21	Alia	39	80000	
22	Sid	41	82000	
21	Abdul	39	58000	

Show 25 per page

## K-Means Algorithm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def kmeans(x, k, max_iters=100):
    centroids = [x[i] for i in range(k)]
    for x in x:
        distances = np.linalg.norm(x - centroids, axis=1)
        cluster_index = np.argmin(distances)
        clusters[cluster_index].append(x)

    new_centroids = []
    for cluster in clusters:
        if cluster:
            new_centroids.append(np.mean(cluster, axis=0))
        else:
            new_centroids.append(centroids[clusters.index(None)])
    if np.allclose(centroids, new_centroids):
        break
    centroids = new_centroids

    return centroids, clusters

data = pd.read_csv(' ')
X = data['values']
k = 3
centroids = np.array(centroids)
```

```
colors =
for i,
for j
plt
CM
import
from
from
import
import
import
this is
X = p
X =
y = p
y = c
model
model
(0,0,0)
plt
```

```
[1] import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

```
[2] df = pd.read_csv('/content/Kmeans_1BM20CS066.csv')
df.head(10)
```

	1	Name	Age	Income(\$)
0	2	Rob	27	70000
1	3	Michael	29	90000
2	4	Mohan	29	61000
3	5	Ismail	28	60000
4	6	Kory	42	150000
5	7	Gautam	39	155000
6	8	David	41	160000
7	9	Andrea	38	162000
8	10	Brad	36	156000
9	11	Angelina	35	130000

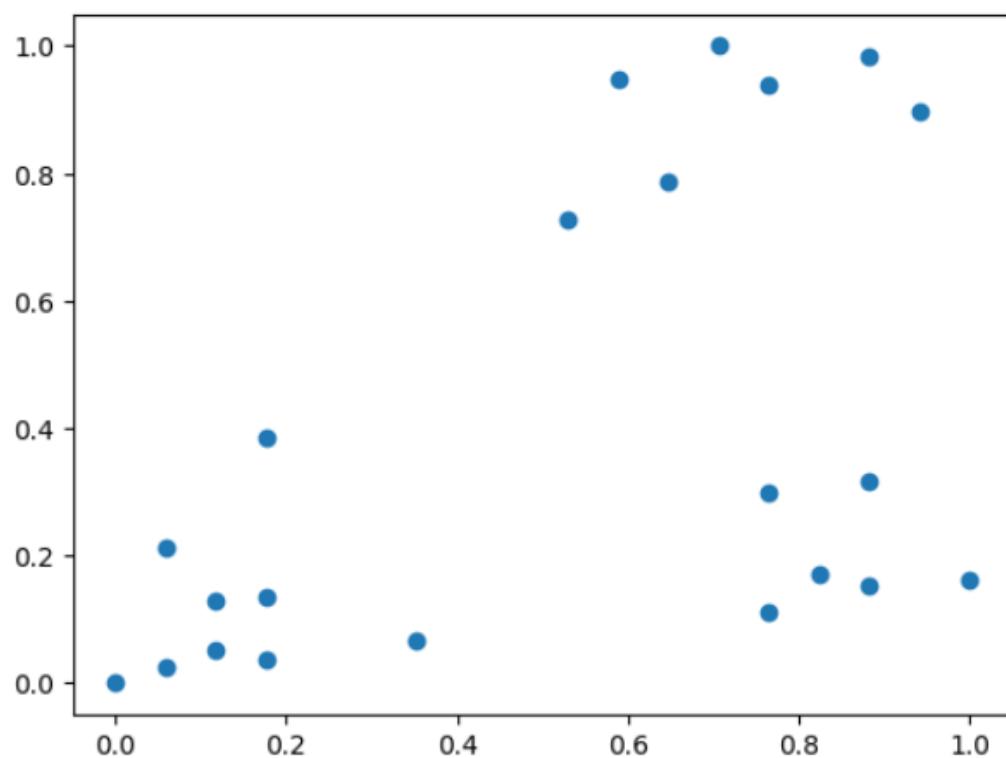
```
[3] scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

	1	Name	Age	Income(\$)
0	2	Rob	0.058824	0.213675
1	3	Michael	0.176471	0.384615
2	4	Mohan	0.176471	0.136752
3	5	Ismail	0.117647	0.128205
4	6	Kory	0.941176	0.897436
5	7	Gautam	0.764706	0.940171
6	8	David	0.882353	0.982906
7	9	Andrea	0.705882	1.000000
8	10	Brad	0.588235	0.948718
9	11	Angelina	0.529412	0.726496

```
▶ plt.scatter(df['Age'], df['Income($)'])
```

```
◀ <matplotlib.collections.PathCollection at 0x7f43820d1a50>
```



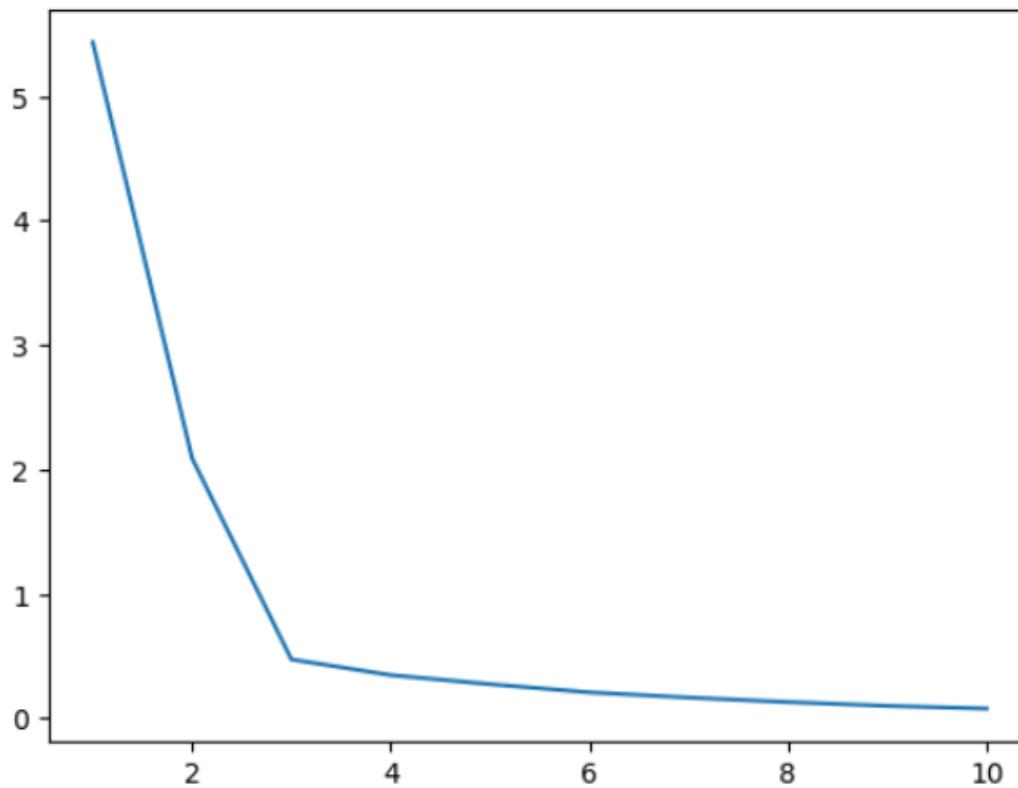
```
▶ k_range = range(1, 11)
```

```
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)
sse
```

```
.....
[5.434011511988178,
 2.091136388699078,
 0.4750783498553096,
 0.3491047094419566,
 0.2798062931046179,
 0.2203764169077067,
 0.1685851223602976,
 0.13265419827245162,
 0.1038375258660356,
 0.08510915216361345]
```

```
plt.xlabel = 'Number of Clusters'  
plt.ylabel = 'Sum of Squared Errors'  
plt.plot(k_range, sse)
```

```
[8] [ <matplotlib.lines.Line2D at 0x7f438004a6e0>]
```



```
[8] km = KMeans(n_clusters=3)  
km
```

```
▼ KMeans  
KMeans(n_clusters=3)
```

```
y_predict = km.fit_predict(df[['Age', 'Income($)']])  
y_predict  
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of  
warnings.warn(  
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2],  
      dtype=int32)
```

```
[10] df['cluster'] = y_predict  
df.head()
```

	1	Name	Age	Income(\$)	cluster
0	2	Rob	0.058824	0.213675	1
1	3	Michael	0.176471	0.384615	1
2	4	Mohan	0.176471	0.136752	1
3	5	Ismail	0.117647	0.128205	1
4	6	Kory	0.941176	0.897436	0

```
[11] df0 = df[df.cluster == 0]  
df0
```

	1	Name	Age	Income(\$)	cluster
4	6	Kory	0.941176	0.897436	0
5	7	Gautam	0.764706	0.940171	0
6	8	David	0.882353	0.982906	0
7	9	Andrea	0.705882	1.000000	0
8	10	Brad	0.588235	0.948718	0
9	11	Angelina	0.529412	0.726496	0

```
✓ [12] df1 = df[df.cluster == 1]
      df1
```

	1	Name	Age	Income(\$)	cluster
0	2	Rob	0.058824	0.213675	1
1	3	Michael	0.176471	0.384615	1
2	4	Mohan	0.176471	0.136752	1
3	5	Ismail	0.117647	0.128205	1
11	13	Tom	0.000000	0.000000	1
12	14	Arnold	0.058824	0.025641	1
13	15	Jared	0.117647	0.051282	1
14	16	Stark	0.176471	0.038462	1
15	17	Ranbir	0.352941	0.068376	1

```
✓ [13] df2 = df[df.cluster == 2]
      df2
```

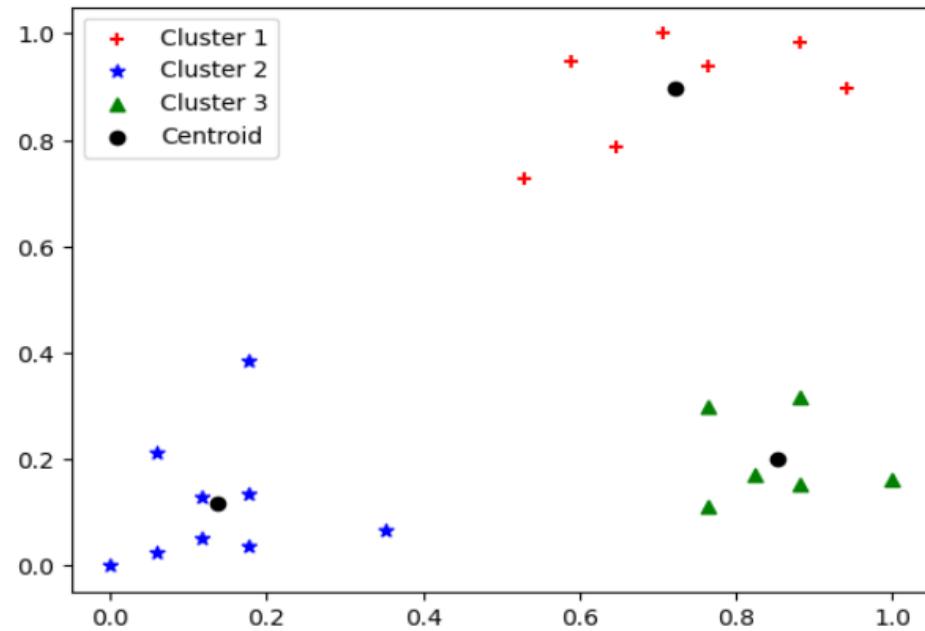
	1	Name	Age	Income(\$)	cluster
16	18	Dipika	0.823529	0.170940	2
17	19	Priyanka	0.882353	0.153846	2
18	20	Nick	1.000000	0.162393	2
19	21	Alia	0.764706	0.299145	2
20	22	Sid	0.882353	0.316239	2
21	21	Abdul	0.764706	0.111111	2

```
✓ [14] km.cluster_centers_
```

```
array([[0.72268908, 0.8974359 ],
       [0.1372549 , 0.11633428],
       [0.85294118, 0.2022792 ]])
```

```
[17] p1 = plt.scatter(df0['Age'], df0['Income($)'), marker='+', color='red')
    p2 = plt.scatter(df1['Age'], df1['Income($)'), marker='*', color='blue')
    p3 = plt.scatter(df2['Age'], df2['Income($)'), marker='^', color='green')
    c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
    plt.legend((p1, p2, p3, c),
               ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

<matplotlib.legend.Legend at 0x7f437d4c73a0>



## Program 8: KNN ALGORITHM

Dataset used: Iris dataset

Algorithm:

- Select the number K of the neighbor
- Calculate the Euclidean distance of K number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category.
- Assign the new data points to that category for which the number of the neighbor is maximum.

K nearest Neighbour

CLASSEmate  
Date: 14/06/23  
Page: \_\_\_\_\_

```
import numpy as np
import os
import fnmatch
import csv
import math

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

def euclidean_distance(instance1, instance2, length):
    distance = 0
    for i in range(length):
        distance += pow((instance1[i] - instance2[i]), 2)
    return math.sqrt(distance)
```

```

def get_neighbours(train_set, test_instance, k):
    distances = []
    length = len(test_instance)
    for train_instance in train_set:
        dist = euclidean_distance(test_instance, train_instance, length)
        distances.append((train_instance, dist))
    distances.sort(key=lambda x: x[1])
    neighbours = []
    for i in range(k):
        neighbours.append(distances[i])
    return neighbours

def predict_class(neighbours):
    class_votes = {}
    for neighbour in neighbours:
        if neighbour[-1] in class_votes:
            class_votes[neighbour[-1]] += 1
        else:
            class_votes[neighbour[-1]] = 1
    sorted_votes = sorted(class_votes.items(), key=lambda x: x[1], reverse=True)
    return sorted_votes[0][0]

def get_accuracy(test_set, predictions):
    correct = 0
    for i in range(len(test_set)):
        if test_set[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0

dataset = load_dataset()

```

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

```

for i in range(4):
    str_column_to_float(dataset, i)

instances, length, dist))

split_ratio = 0.7
train_size = int(len(dataset) * split_ratio)
train_set = dataset[:train_size]
test_set = dataset[train_size:]

k = 3
predictions = []
for test_instance in test_set:
    neighbours = get_neighbours(train_set, test_instance, k)
    predicted_class = predict_class(neighbours)
    predictions.append(predicted_class)
    print('Expected:', test_instance[-1], 'Predicted:', predicted_class)

accuracy = get_accuracy(test_set, predictions)
print('Accuracy', accuracy)

OUTPUT:
Expected: setosa Predicted: setosa
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica

```

✓

```
0s  import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
|
def most_common(lst):
    return max(set(lst), key=lst.count)

def euclidean(point, data):
    # Euclidean distance between points a & data
    return np.sqrt(np.sum((point - data)**2, axis=1))

class KNeighborsClassifier:
    def __init__(self, k=5, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
```

```
✓ 0s
def evaluate(self, X_test, y_test):
    y_pred = self.predict(X_test)
    accuracy = sum(y_pred == y_test) / len(y_test)
    return accuracy

iris = datasets.load_iris()
X = iris['data']
y = iris['target']

# Split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Preprocess data
ss = StandardScaler().fit(X_train)
X_train, X_test = ss.transform(X_train), ss.transform(X_test)

# Test knn model across varying ks
accuracies = []
ks = range(1, 30)
for k in ks:
    knn = KNeighborsClassifier(k=k)
    knn.fit(X_train, y_train)
    accuracy = knn.evaluate(X_test, y_test)
    accuracies.append(accuracy)
# Visualize accuracy vs. k
fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="k",
       ylabel="Accuracy",
       title="Performance of knn")
plt.show()
```

**Program 9:** Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Algorithm for k means clustering:

- Initialize k means with random values
- For a given number of iterations:
- Iterate through items:
- Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
- Assign item to mean
- Update mean by shifting it to the average of the items in that clusters

Algorithm for EM algorithm:

- The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
- This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- This step is known as Maximization or M-step, where we use complete data obtained from the 2<sup>nd</sup> step to update the parameter values. Further, M-step primarily updates the hypothesis.
- The last step is to check if the values of latent variables are converging or not.

Date \_\_\_\_\_  
Page \_\_\_\_\_

23

```

colors = ['r', 'g', 'b']
for i, cluster in enumerate(clusters):
    for point in cluster:
        plt.scatter(point[0], point[1], c=colors[i])
plt.show()

```

→

void

EM

```

import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

```

`iris = datasets.load_iris()`

`((clusters))`

```

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length',
             'Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

```

```

model = KMeans(n_clusters=3)
model.fit(X)

```

```

colormap = np.array(['red', 'lime', 'black'])

```

```

plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets],
            s=60)

```

Page \_\_\_\_\_

```

plt.title('K-Means Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Means:', sm.accuracy_score(y, model.labels_))
print('The confusion matrix of K-Means:', sm.confusion_matrix(y, model.labels_))

```

`from sklearn import preprocessing`

`scaler = preprocessing.StandardScaler()`

`scaler.fit(x)`

`X_S = scaler.transform(x)`

`df_S = pd.DataFrame(X_S, columns=x.columns)`

`df_S.sample(5)`

`from sklearn.mixture import GaussianMixture`

`gmm = GaussianMixture(n_components=3)`

`gmm.fit(x)`

```

plt.subplot(2, 2, 3)
plt.scatter('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

```

```

print('Accuracy of EM:', sm.accuracy_score(y, gmm.predict(X_S)))
print('Confusion Matrix:', sm.confusion_matrix(y, gmm.predict(X_S)))

```

## Dataset: Iris dataset

```
✓ 2s import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])
```

```
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm
```

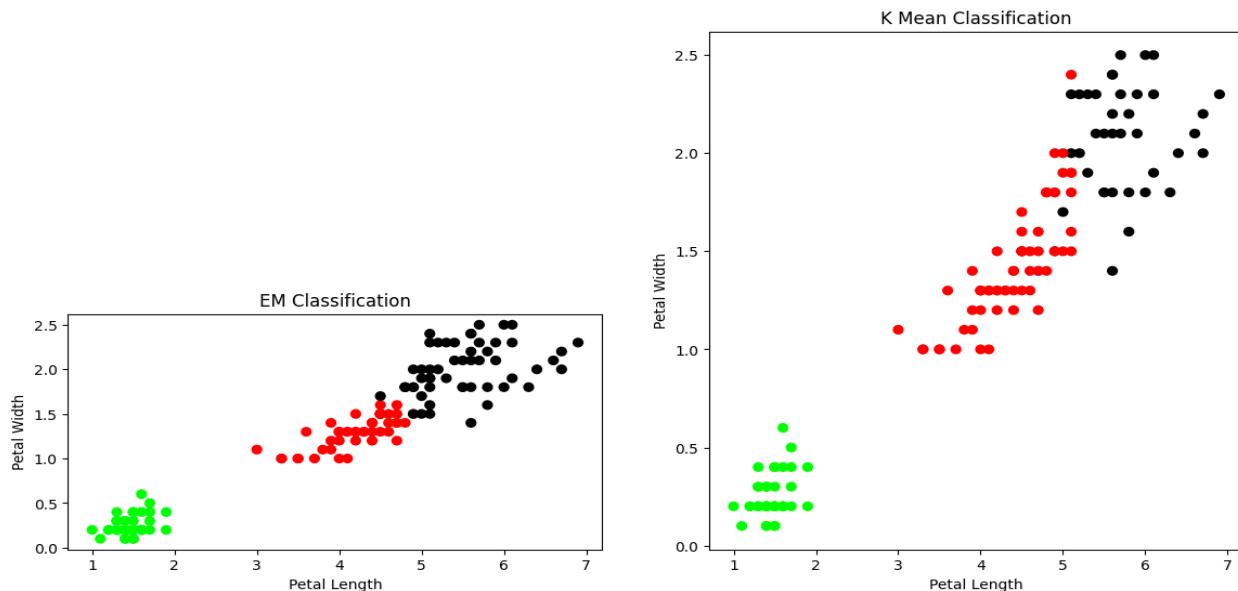
```

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('EM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))

```

The accuracy score of K-Mean: 0.24  
The Confusion matrix of K-Mean: [[ 0 50 0]  
[48 0 2]  
[14 0 36]]  
The accuracy score of EM: 0.3333333333333333  
The Confusion matrix of EM: [[ 0 50 0]  
[45 0 5]  
[ 0 0 50]]



**Program 10:** Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

Algorithm:

- F is approximated near  $x_q$  using a linear function:

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

- Minimize the squared error:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

- It is weighted because the contribution of each training example is weighted by its distance from the query point.

The image shows handwritten notes on a piece of lined paper. At the top left, it says "Locally weighted regression". Below that is a Python code snippet. The code imports numpy, bokeh, and bokeh.io. It defines a function `local_regression` that takes `x0, X, Y, tau` as parameters. Inside, it calculates `X` as a column of ones and `X.T`, then uses `linalg.inv(X.T @ X) @ X @ Y` to find `beta`. It also defines a `radial_kernel` function that returns `np.exp(-sum((X - x0)**2, axis=1) / (-2 * tau * tau))`. The variable `n` is set to 1000. The code then generates data sets `X` and `y`, normalizes `X`, and creates a plot titled "tau vs y.tau" with a scatter plot of `x, y` and a prediction line. The notes are dated "Date: 14/10/23".

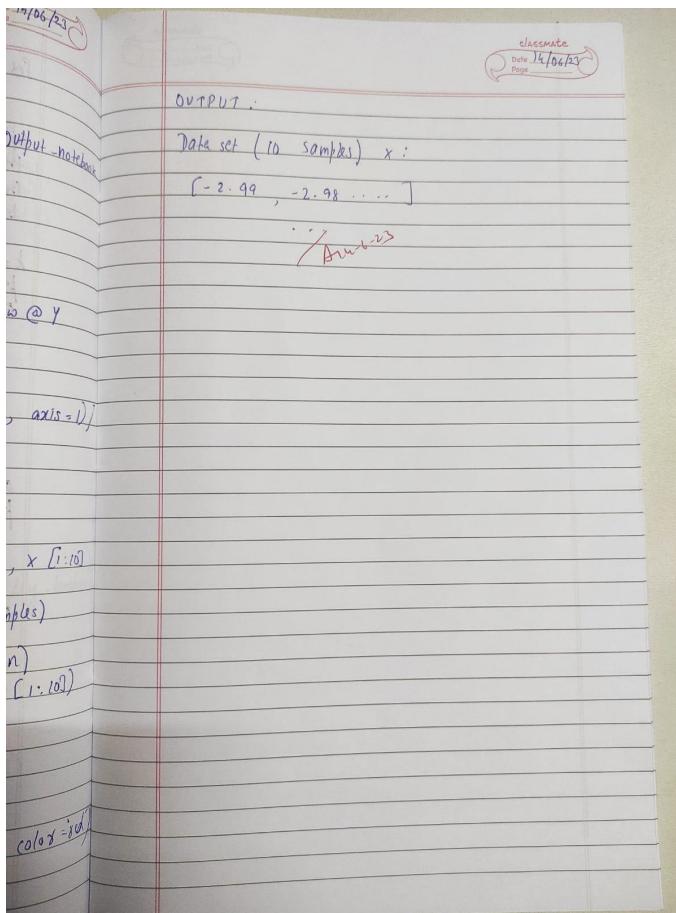
```

import numpy as np
from bokeh import plotting, figure, show, OutputFileServer
from bokeh.layouts import gridplot
from bokeh.io import push_notebook
def local_regression(x0, X, Y, tau):
    X = np.c_[np.ones(len(X)), X]
    X_T = X.T
    beta = np.linalg.inv(X_T @ X) @ X @ Y
    return x0 @ beta

def radial_kernel(x0, x, tau):
    return np.exp(-sum((x - x0)**2, axis=1) / (-2 * tau * tau))

n = 1000
X = np.linspace(-3, 3, num=n)
print("The data set (10 samples) X : \n", X[:10])
y = np.log(np.abs(X**2 - 1) + 5)
print("The fitting (true) data set (10 samples) Y : \n", y[:10])
X += np.random.normal(scale=0.1, size=n)
print("Normalised (10 samples) X : \n", X[1:10])
domain = np.linspace(-3, 3, num=300)
def plot_lwr(tau):
    plot.title.text = 'tau vs y.tau'
    plot.scatter(x, y, alpha=.8)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

```



Dataset: tip.csv

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

[ ] def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

[ ] def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
```

```
def localWeightRegression(xmat, ymat, k):
    m, n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i], xmat, ymat, k)
    return ypred
```

```
[ ] def graphPlot(X, ypred):
    sortindex = X[:,1].argsort(0)
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill, tip, color='green')
    ax.plot(xsort[:,1], ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
```

```
data = pd.read_csv('/content/tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```